TracHack 21.2- Predicting Upgrades Background

T1 2021

Group：HD group

Mintao Yi z5233399

Xin Cheng z5231019

Tong Zhang z5235242

Yuhao Zhou z5227282

Chengdong He z5285256

Introduction

This paper aims at developing a machine learning method to predict whether TracFone users intend to upgrade their devices or not. The proposed techniques include two major stages: data cleaning and machine learning prediction. A method of ensemble learning is used as final classifier to achieve a high accurate prediction.

TracHack Challenge 21.2 is organized by TracFone, a pioneer in US telecommunications field. The goal is to develop an advanced prediction method on users' intention of device upgrade. With accurate prediction results, TracFone can expanse business and deliver exact device upgrade information to users.

The TracFone dataset includes several csv files, including basic user information, carrier information, contract-information, network usage information, etc.

The approach contains data cleaning, data preprocessing, model selection and tuning parameters. Data cleaning was the first step, in order to retain the data relevant to user upgrade. Secondly, data was processed into a uniform format for model training. Thirdly, three type of ensemble learning classifiers were applied with parallel competition: `GradientBoostingDecisionTreeClassifier`, `RandomForestClassifier`, `XGBClassifier`. Finally, `XGBoost` was selected as the final classifier and `GridSearchCV` method was used to choose the optimal parameters. Finally, the prediction results were compared with the ground truth value, and the performance of the models was evaluated by quantitative criteria such as f1_score, accuracy and visualization results.

Exploratory Data Analysis

The data used in this paper was the TracFone dataset, which includes several csv files as `upgrades.csv`, `customer_info.csv`, `suspensions.csv`, `redemptions.csv`, `reactivations.csv`, `deactivations.csv`, `phone_info.csv`, `lrp_enrollment.csv`, `lrp_points.csv` and `network_usage_domestic.csv`. Table 1presents the basic information and Table 2 presents the features contained in each csv file.

CSV files	Contents
upgrades.csv	Base dataset that has line_id, upgrade_date and upgrade columns
customer_info.csv	Customer info has carrier, plan and activation information for each line_id
phone_info.csv	Phone info has all the device information for each line_id
redemptions.csv	Redemptions has all the plan redemption details for each line_id
deactivations.csv	Deactivations has the deactivation details for each line_id
reactivations.csv	Reactivations has the reactivations details for each line_id
suspensions.csv	Suspension is when a customer is more than 15 days past due.
network_usage_domestic.csv	Domestic network usage has the network usage details for each line_id
lrp_points.csv	Lrp points has the loyalty reward details for each line_id.
lrp_enrollment.csv	Lrp enrollment has the loyalty reward enrollment details for each line_id

Table 1

CSV files	Features					
upgrades.csv	line_id	date_observed	upgrade			
customer_info.csv	line_id	carrier	first_activation_date	plan_name	plan_subtype	redemption_date
phone_info.csv	line_id	cpu_cores	expandable_storage	gsma_device_type	gsma_model_name	gsma_operating_system
		lte	internal_storage_capacity	lte_advanced	lte_category	manufacturer
		os_family	os_vendor	os_version	sim_size	total_ram
		touch_screen	wi_fi	year_released		
redemptions.csv	line_id	channel	gross_revenue	redemption_date	redemption_type	revenue_type
deactivations.csv	line_id	deactivation_date	deactivation_reason			
reactivations.csv	line_id	reactivation_channel	reactivation_date			
suspensions.csv	line_id	suspension_start_date	suspension_end_date			
network_usage_domestic.csv	line_id	date	hotspot_kb	kb_5g	mms_in	mms_out
		sms_in	sms_out	total_kb	voice_count_in	voice_count_total
		voice_min_in	voice_min_out			
lrp_points.csv	line_id	quantity	status	total_quantity	update_date	
lrp_enrollment.csv	line_id	lrp_enrolled	lrp_enrollment_date			

Table 2

These csv files contain 10 classes and 69 features. Each of these classes uses `line_id` as the primary key. We analyzed these features and found that certain features were correlated. For example, `redemption_date` in `customer_info` was associated with the feature `redemption_date` in the redemptions class. Figure 3 represents the combination of features with correlation.

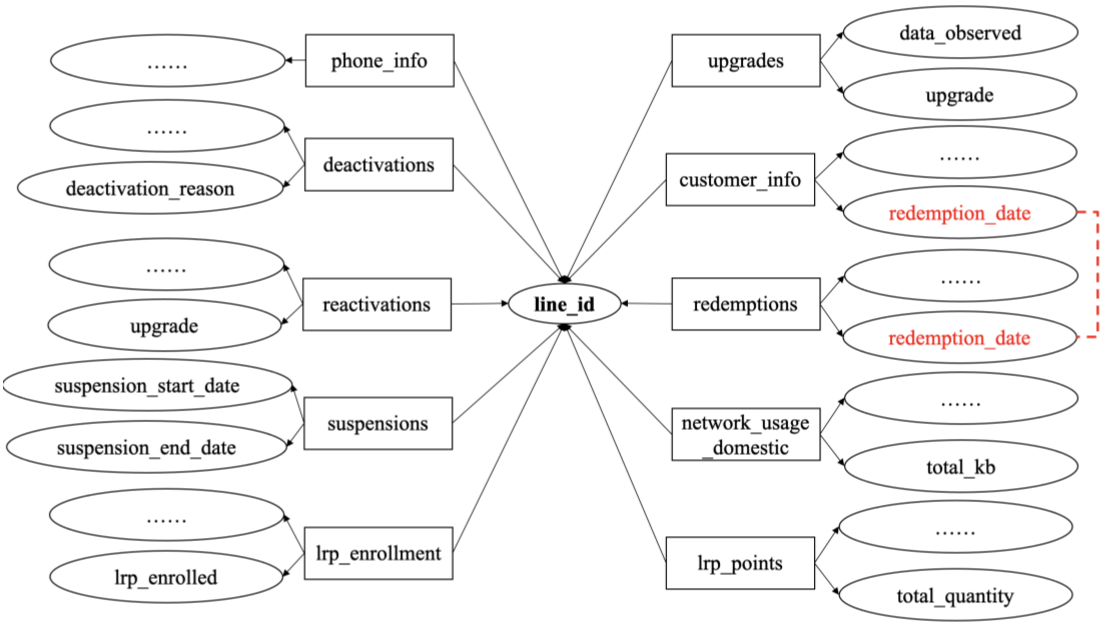

Figure 1

Figure 1 shows the one-to-many feature combinations in suspensions class and acitivation class. Some of these features are irrelevant to make the prediction whether the user would update the device, so we filtered the features based on analysis and experience but kept the useful features. The specific data filtering methods will be shown in Section 3.

Methodology

The pipeline of the methodology developed for this specific project is shown in Figure 2. In this part, the detailed presentation of the applied method would be introduced and justified, based on the pipeline.

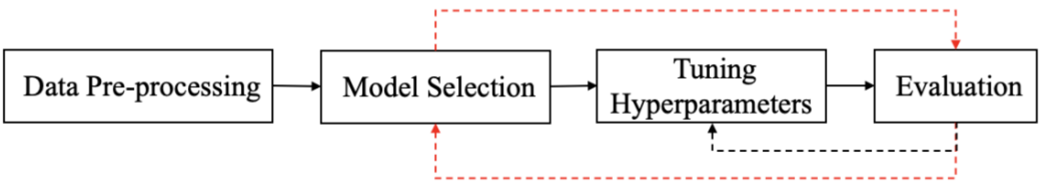

Figure 2

The pipeline consists of four parts. The preprocessing of data was applied firstly, then different models would be trained immediately (without tuning the hyperparameters). The most suitable model would be selected after evaluating the performance. The hyper-parameters of this selected model would be tuned, and the performance of the model will be evaluated. In addition, necessary fine-tuning to the hyperparameters would be applied repeatedly, until the performance of the model became satisfying.

The next few sections will focus on the detailed implementation of each step in the pipeline, including the specific tasks in each step('What?'), why these tasks are indispensable('Why?'), and the methods applied('How?').

4.1 Data Pre-processing

As explained in introduction, the data for this project was complex, with a large size. The usability of the data was not guaranteed, and the data was very fragmented. To have a better understanding of the data, as well as obtain the data that would be easier to be used for the model, normal methods for data pre-processing were applied.

4.1.1 Encoder for data

Generally, machine learning prefers numeric data, which is more convenient to make calculation and improve the efficiency. Thus, we did necessary encoder for non-numeric data.

`LabelEncoding` was applied for category variables, such as `'cpu_cores'`, `'os_name'`, etc. Time variables were converted into a number by `time.mktime()` and then normalized. Normalization could improve the efficiency of model training and improve the accuracy [1].

One-Hot encoder was applied for features 'deactivation_reason_ACTIVE UPGRADE', 'deactivation_reason_UPGRADE' and 'deactivation_reason_PASTDUE', to make the distance computational.

4.1.2 Handling missing values and outliers

The missing values for time features were filled with the mean (of the normalized value), but those for other numeric attributes were filled with -1, i.e. assigned a new category to the missing values. This approach is easy to implement. Compared with deleting data with missing values, it could also avoid wasting the information in other attributes. Especially considering the number of attributes was relatively high, deleting the data with missing values immediately could be extremely unworthy.

4.1.3 Feature Extraction

It should be noticed that there were significantly large number of features. This tended to make the calculation become complex and make the speed of training slow. On the other hand, not all features were informative for making the prediction, thus may result in overfitting. Therefore, feature extraction would be essential.

Analysis based on experience, Stability Selection and `xgboost.feature_importance` were three methods applied to extract features in this specific project[2]. Features that regarded as useful were selected and those regarded as useless were abandoned, according to not only the experience but also the analysis of data. Then stability selection was applied to verify the previous selection, by comparing the importance based on the times that features were selected as important. Finally, `xgboost` provides a function that could list the importance of different features, based on which we made a further extraction.

Eventually, only 32 features would be taken into consideration.

4.2 Model Selection

This project could be considered as a typical binary classification problem. Due to the big size of the data, `MNB` might be unbefitting for this task. We compared the performance of `SVM, Random Forest, Gradient Boosting Decision Tree` and `XgBoost` respectively, without tuning the hyper-parameters. Based on their performance (Table 3), `XgBoost` was selected as the most suitable model.

Model	F1-Score	Precision	Recall	Accuracy
SVM	0.51	0.69	0.67	0.79
RF	0.83	0.87	0.88	0.92
GBDT	0.85	0.85	0.90	0.92
XGB	0.85	0.85	0.92	0.93

Table 3

`XgBoost` adds the regular terms to control the complexity of the model, which could avoid overfitting. In addition, `XgBoost` ensures proper decoupling due to its more flexible loss function [3].

4.3 Tuning Hyper-parameters

`GridSearchCV` was used repeatedly to tune the hyperparameters. It uses grid search to find the best parameters based on the given scoring function.

For `XgBoost`, `'max_depth'` and `'min_weights'` control the generation of the tree, thus have the most important influence in the performance [4,5]. These two hyperparameters were tuned firstly, and others were tuned then. The final hyperparameters for the model are listed below:

```
(n_estimators=6000,learning_rate=0.01,max_depth=10,min_child_weight=1,subsample=0.9,gamma=0.2,colsample_bytree=0.7,objective='binary:logistic',nthread=4,scale_pos_weight=1,seed=27,reg_alpha=1e-05, use_label_encoder = False)
```

4.4 Evaluation

Using the trained model with tuned hyperparameters, the prediction of the test data set could be made then. In this step the evaluation for the performance of the model was made, based on the prediction.

Specifically, Precision, Recall, F1-score, Accuracy and Roc-Auc-score were taken into consideration.

For this specific task, F1-score was the main criteria to evaluate the model. However, the others were also considered.

Results

5.1 Result and Metrics

After tuning the hyper-parameters, the f1-score of the model (`Xgboost`) was 0.88 on the test set.

In order to have a more complete evaluation of the performance, more metrics other than f1-score were considered to measure the models, including time, recall, precision and accuracy. Precision reflects the model's ability in predicting TP samples, while recall reflects how accurately our model is able to identify the data. F1 score takes the consideration of both precision and recall.

$$recall = \frac{tp}{tp + fn}$$

$$Precision = \frac{tp}{tp + fp}$$

$$Accuracy = \frac{tp + tn}{tp + tn + fp + fn}$$

Model	Multilayer Perceptron	XGBoost	Gradient Boost Decision Tree	RandomForest	SVC
Time	95.97s	340.46s	128.74s	13.73s	64.92s

Table 4

Based on Table 4 and Figure 3, `XGBoost` cost the most time, but performed the best in other four metrics (f1-score, accuracy, recall and precision). According to the graph, if the time efficiency would be considered very important, `GBDT` will be a good choice worthy consideration, since it had the similar performance as `XGBoost` but cost less time.

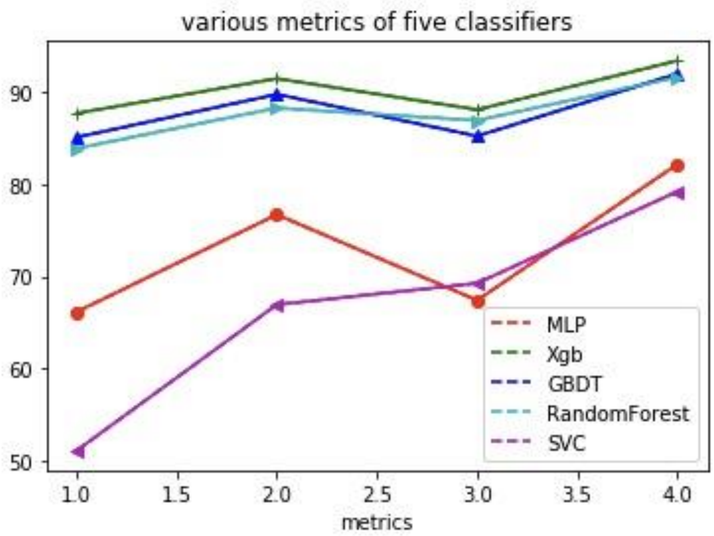

Figure 3

The highest f1-score of XgBoost means that XgBoost has relative better performance on both precision and recall simultaneously, as well as the discrepancy of both being as little as possible. Based on the metrics, it is reasonable to make the conclusion that the performance of XgBoost is satisfying for this specific project.

5.2 Design Choice

Based on the performance in different metrics, we choose `XGBoost` as our final model. There are several essential hyper-parameters in `XGBoost`, like `n_estimators`, `learning_rate`, `min_child_weight`, `max_depth`, `gamma`, `subsample`, `objective`, `seed`, `reg_alpha`, `colsample bytree`, `nthread`, `scale_pos_weight`, `seed`, `use_label_encoder`. `n_estimators` is the number of gradient boosted trees. `Max_depth` represents the maximum tree depth for base learners. `Learning rate` is the Boosting learning rate. `Objective` specifies the learning task and the corresponding learning objective or a custom objective function to be used. `Gamma` is minimum loss reduction required to make a further partition on a leaf node of the tree. `Min_child_weight` is minimum sum of instance weight(hessian) needed in a child. `Reg_alpha` is L1 regularization term on weights. `Colsample_bytree` is the subsample ratio of columns when constructing each tree. `Scale_pos_weight` reflects the balancing of positive and negative weights.

In order to get the best parameters, GridSearchCV was applied to tune parameter. The parameters were optimized by cross-validate grid-search over a parameter grid, and finally got the best parameter set.

5.3 Feature Importance

`XgBoost.feature_importance` was adopted to help to recognize the importance of features and extract informative features. Based on Figure 4, carrier (of the line) was the most important feature base on the average gain of splits. Redemption date (date on which the line redeemed the current plan) also had significant effect on making the prediction.

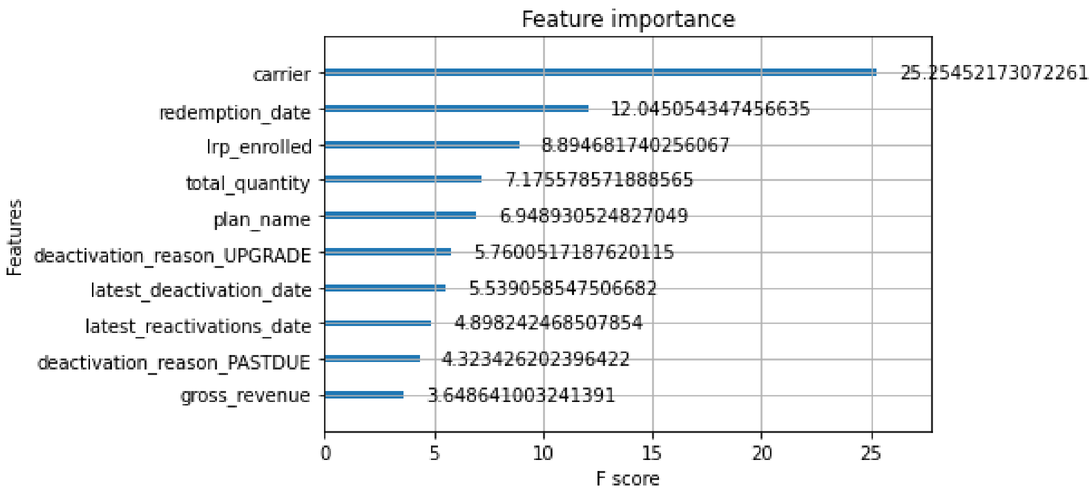

Figure 4

Discussion

6.1 Comparison of different models

Table 5 analyses different models adopted in the experiment, in terms of the theory and time complexity:

Model	Description of principle	Time Complexity
Multilayer Perceptron	Multi-layer perceptron solves the problem of linear insepara bility by adding a hidden layer between input and output to improve the efficiency.	O(m*n^2) M samples
Random Forest	Random forest is a kind of integration model, which adopts the general method of Bootstrap aggregation as its training algorithm. Compared with decision tree, it can prevent overfitting.	O(m*n*logn). N samples, M features
SVM	SVM is a dichotomous model that maps the eigenvectors of an instance to points in space and "best" distinguishes the two categories by drawing a line. SVM is suitable for small and medium-sized data samples, nonlinear, high-dimensional classification problems.	O(n*k) K features dimension
GBDT	Through multiple rounds of iteration, GBDT generates a weak classifier in each iteration, and each classifier is trained on the basis of the gradient of the previous classifier. The training process is to continuously improve the accuracy of the final classifier by reducing the deviation.	O(n*logn * d * m) N samples, D features, M is the depth of the tree
XGBoost	XGBoost has similar performance to GBDT. It adds regular terms to control the complexity of the model, which is beneficial to prevent overfitting and improve the generalization ability of the model. It also automatically learns the processing strategy for missing values.	O(n*d*k*logn) D features, K is the depth of the tree

Table 5

Since the ranking of the competition was based on the F1_Score metric, the evaluation of models could be shown as follows (" > "represents the more suitable model):

`XGBoost > GBDT > Random Forest > MLP > SVM`

`XGBoos`t had the best performance for this specific project. As explained in Methodology, `XgBoost` adds the regular terms to control the complexity of the model, which could avoid overfitting. In addition, `XgBoost` ensures proper decoupling due to its more flexible loss function.

The low time efficiency of XgBoost may be related to its theory. Firstly, the splitting of nodes requires to traverse the whole data set. Secondly, the pre-sort consumes much memory space, since not only the characteristic value but also the statistic of corresponding gradient needs to be stored.

6.2 Metric Evaluation

In order to comprehensively evaluate the performance of our model for the project. F1-score, time, recall rate, accuracy and accuracy were all taken into consideration (Table 6).

Metrics	Multilayer Perceptron	XGBoost	Gradient Boost Decision Tree	RandomForest	SVC
F1-Score	0.66	0.88	0.85	0.83	0.51
Precision	NULL	0.88	0.85	0.87	0.69
Recall	NULL	0.92	0.90	0.88	0.67
Accuracy	NULL	0.93	0.92	0.92	0.79
Time	95.97s	340.46s	128.74s	13.73s	64.92s

Table 6

As mentioned in part of "Result", we finally found that although `XGBoost` had the highest time cost, it had the best performance in the four indexes of F1, accuracy, recall rate and precision. The time cost of `GBDT` is one third of that of `XGBoost`, but its performance in the other four indicators is lower than that of XGBoost. The time cost of `Random Forest` was the least, and its comprehensive performance was the third. The time cost of MLP and SVM is not high, but their performance is not good. Therefore, `GBDT` or `Random Forest` can be selected if time efficiency is preferred. However, `GBDT` model is more recommended compared with `Random Forest`, because the performance of `GBDT` and `XGBoost` model were similar. Due to the specific requirements of the project (considering f1-score only), the final optimal model was `XGBoost`.

6.3 Future Improvement

Since the project is in the form of competition, there was limited time to carry out experiments and completed the project. In terms of data processing and ensemble methods, we believe that there can be further improvement in future experiments.

In terms of data processing, there are the following considerations:

1. For a table with a one-to-many relationship, is it reasonable to select only the latest date as the feature? In addition to the number of times, does the interval between each

activation need to be considered?

2. For null value processing, whether filling -1 is reasonable, and whether filling the average value or mode value will improve the experimental results?

3. Whether the selection of features is reasonable depends solely on human experience? The distribution of labels can be viewed through the one-to-one connection between features and labels.

4. Build better neural networks to learn features.

The ensemble methods are the techniques of creating multiple models and then combining them to produce improved results. It includes voting, stacking, bagging and boosting. The `XGBoost, GBDT, Random Forest`, etc. in this experiment were input into the ensemble method as the basic models. The same training data set and different segmentation of the same algorithm were used to create each basic model, or the same data set with different algorithms or any other method was used to create each basic model. Through boosting algorithm can improve the classification performance by changing the weight of the samples, learning multiple classifiers and combining these classifiers. The above ideas can be tried in future experiments to improve the prediction results of experimental data.

Conclusion

The purpose of the experiment of this project is to predict in advance based on the feedback information of users, so as to judge whether users will upgrade their communication equipment. Based on this goal, we will make predictions based on a variety of data to determine whether users will choose to upgrade their communication devices. The f1-score of the model developed in this project was 0.88.

As for the experimental method, data preprocessing (data encoder, handling missing values and outliers, features extraction) was carried out first, and then model selection was performed. The optimal parameter setting was found by `GridSearchCV` approach, and the optimal model (`XGBoost`) of this experiment was determined by the performance score of F1_score. In the result section, the test extends the metrics with time, recall, precision and accuracy for the experimental model. The performance of the model can be evaluated more comprehensively by adding these metrics. We believe that the experimental results can be further optimized in data processing and ensemble methods in the future. In this way, more accurate prediction data can be obtained to determine whether users will choose to upgrade their communication equipment, so as to ensure that only those customers who want to upgrade their equipment can really get the opportunity to upgrade their equipment and provide better service experience for users.

Reference

1. Alam, M.J., Ouellet, P., Kenny, P. and O'Shaughnessy, D., 2011, November. Comparative evaluation of feature normalization techniques for speaker verification. In International Conference on Nonlinear Speech Processing (pp. 246-253). Springer, Berlin, Heidelberg.

2. Guyon, I., Gunn, S., Nikravesh, M. and Zadeh, L.A. eds., 2008. Feature extraction: foundations and applications (Vol. 207). Springer.

3. Chen, T. and Guestrin, C., 2016, August. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794)

4. https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/.

5. Putatunda, S. and Rama, K., 2018, November. A comparative analysis of hyperopt as against other approaches for hyper-parameter optimization of XGBoost. In Proceedings of the 2018 International Conference on Signal Processing and Machine Learning (pp. 6-10)